Ayhem Bouabid DS-BS-20 a.bouabid@innopolis.university

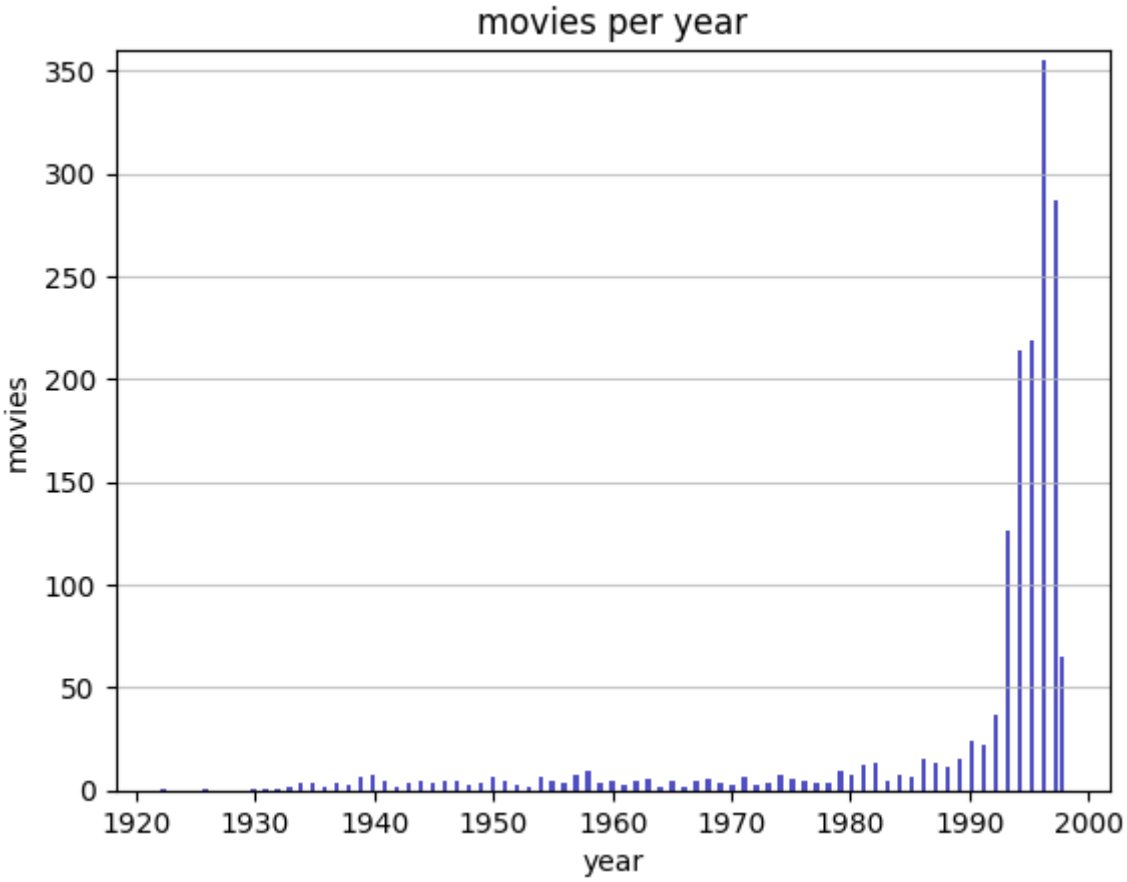
Introduction

Data analysis

Movie Data

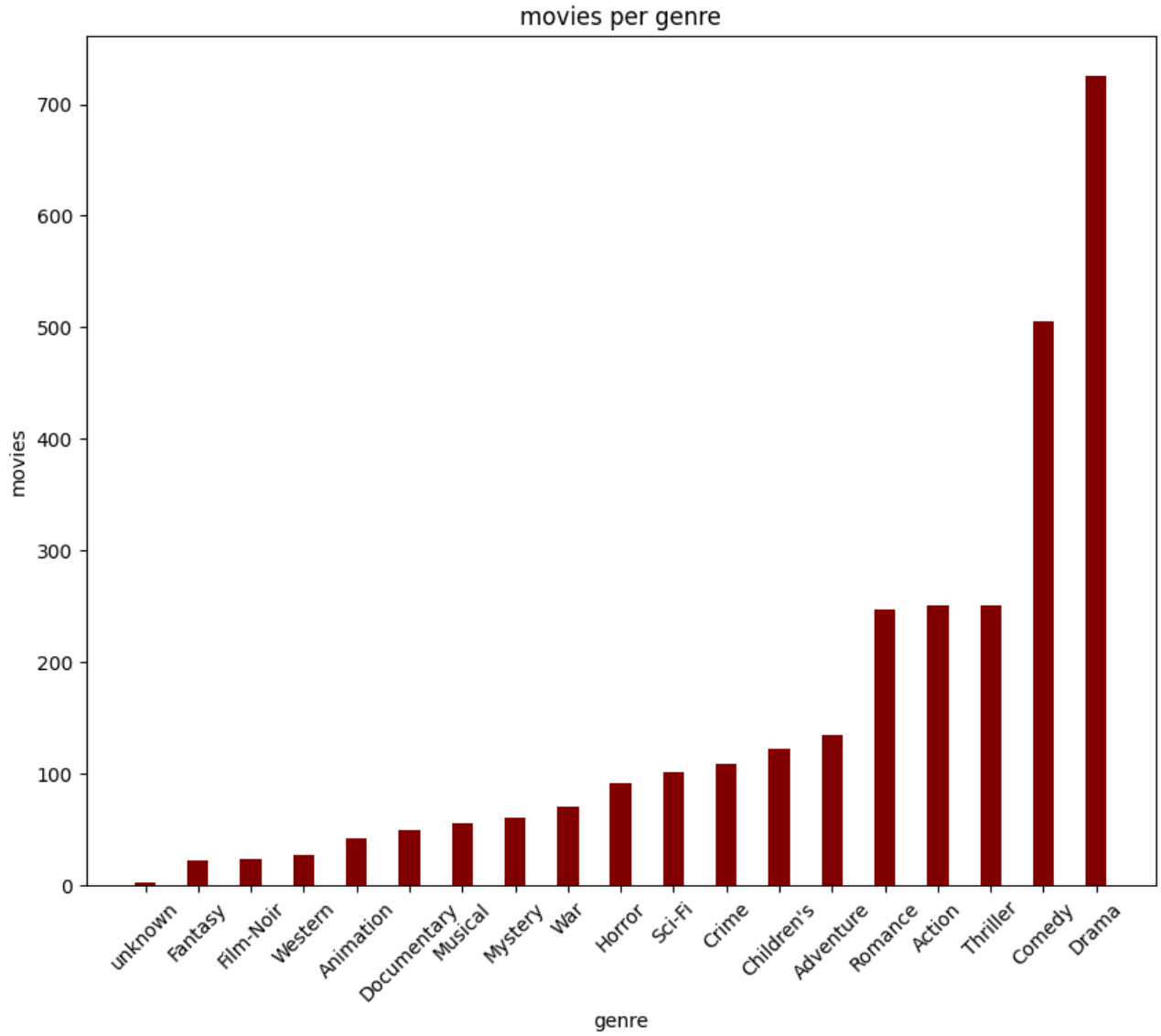
A movie is associated with 24 initial columns: id, title, link to the imdb page, the release data and 19 binary columns that represent different genres.

1. The title information might be quite misleading by itself. Completely movies can be assigned very similar weights. Thus, without further context such as description, or short movie summary this field might do more harm than good
2. The link to the imdb page is definitely helpful for data mining purposes. However, imdb is taking extra measures against web scraping making it quite difficult to extract information for all the entries in the dataset (it would be extremely suspicious for a human user to read about a movie released at 1995 just to check out a movie released in 1935 few minutes later). Thus, both these fields are dropped
3. The date is reduced to the year
4. We can see that the majority of movies were released between the year 1990 and 2000. The latter might lead to a bias towards more recent movies

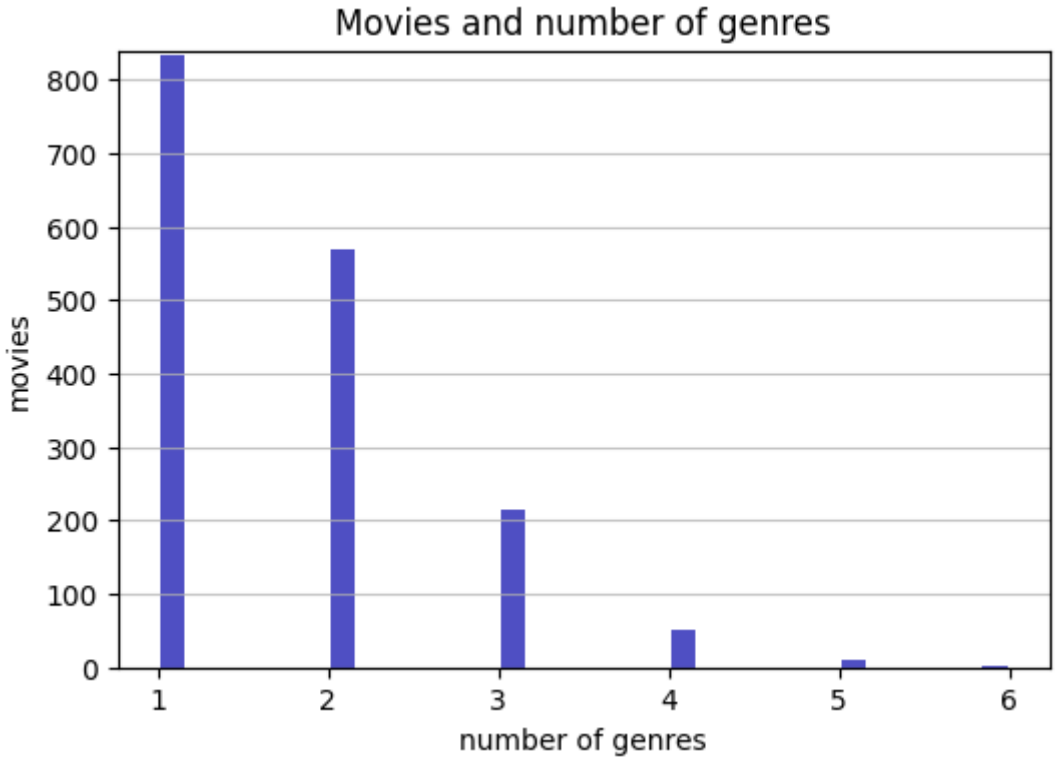


5. A couple of interesting remarks can be made about the genre columns:

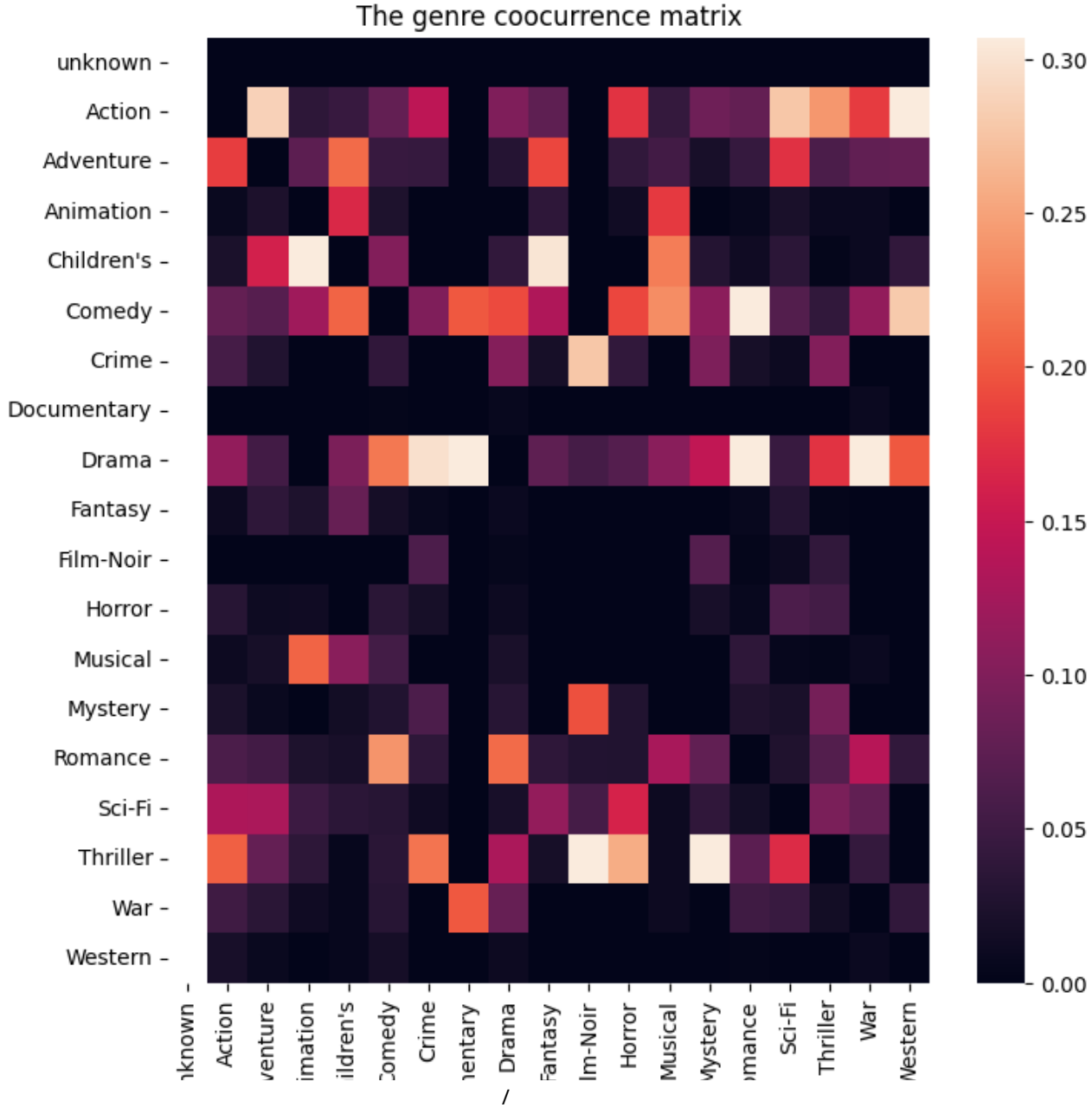
    1. Genre Inbalance:



    2. Sparsity: only $5%$ of movies have more than 4 genres:


Movies and number of genres
movies
number of genres
0
100
200
300
400
500
600
700
800
1
2
3
4
5
6

A | B |


The genre coocurrence matrix
unknown
Action
Adventure
Animation
Children's
Comedy
Crime
Documentary
Drama
Fantasy
Film-Noir
Horror
Musical
Mystery
Romance
Sci-Fi
Thriller
War
Western
0.30
0.25
0.20
0.15
0.10
0.05
0.00

/

The combination of these 2 figures suggests that Even though only 216 (please check the movie_data_analysis notebook for the code) out of 2^{19} possible combinations are present, the combinations are still diverse (we cannot manually determine the major combinations and limit the cardinality of this categorical field)

Therefore, the interactions between the different genres are:

- too complex to be significantly improved with feature engineering
- too sparse in consideration of the total number of features.

It is known that content-based recommendation systems offers several advantages such as nich recommendataions, scalability and simpliticiy. Nevertheless, the success of such an approach heavily depends on the quality of the item representations. In our case the item features are unlikely to be expressive enough.

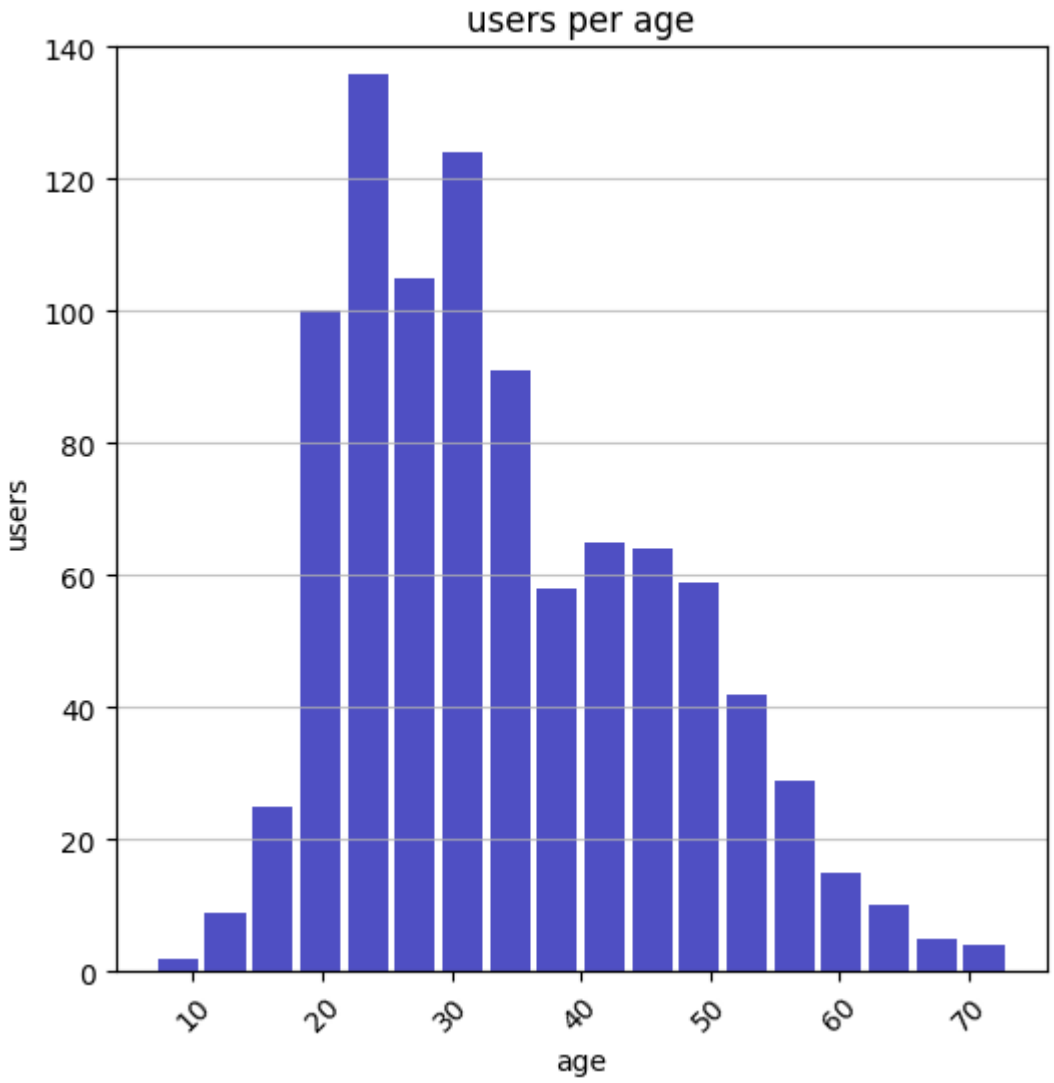
Building a good-performing Recommendation system on this dataset would require Collaborative Filtering.

Analysing the user data

User is associated with 4 fields:

1. id
2. age
3. gender
4. job
5. zip_code

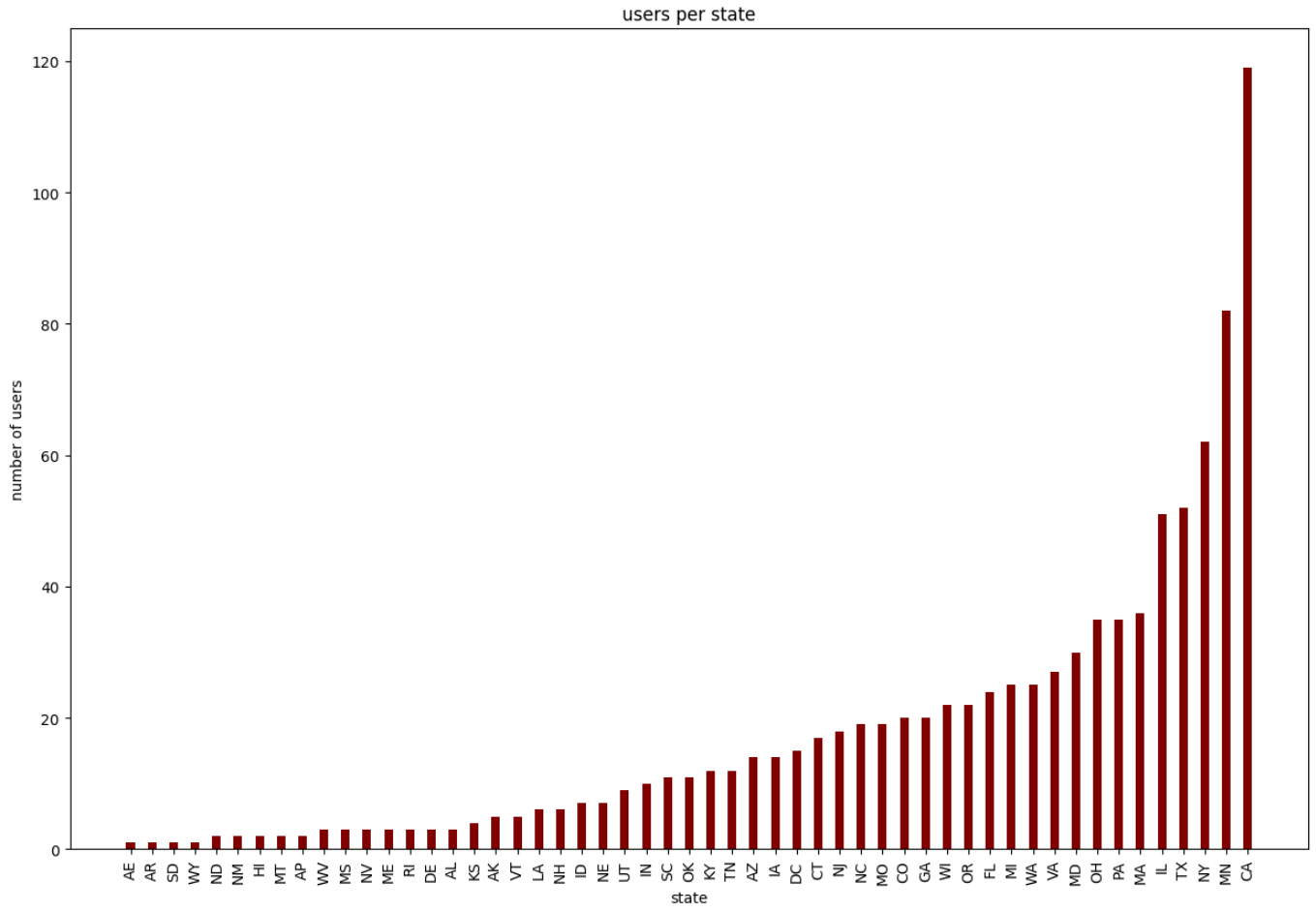
It was interesting to consider the demographics of our users: The age distribution is quite similar to the Gaussian distribution:



This is quite promising since Normal distribution is known for its desirable statistical properities and the data can be converted to Standard Distribution by using scaling.

zip_code was considered a slightly problematic column due to its large number of unique values without inherent direct correlation to the user's movie taste. Thus, the first step was to extract more detailed

information: the state. Even the *'state'* variable still had very skewed distribution as displayed below:



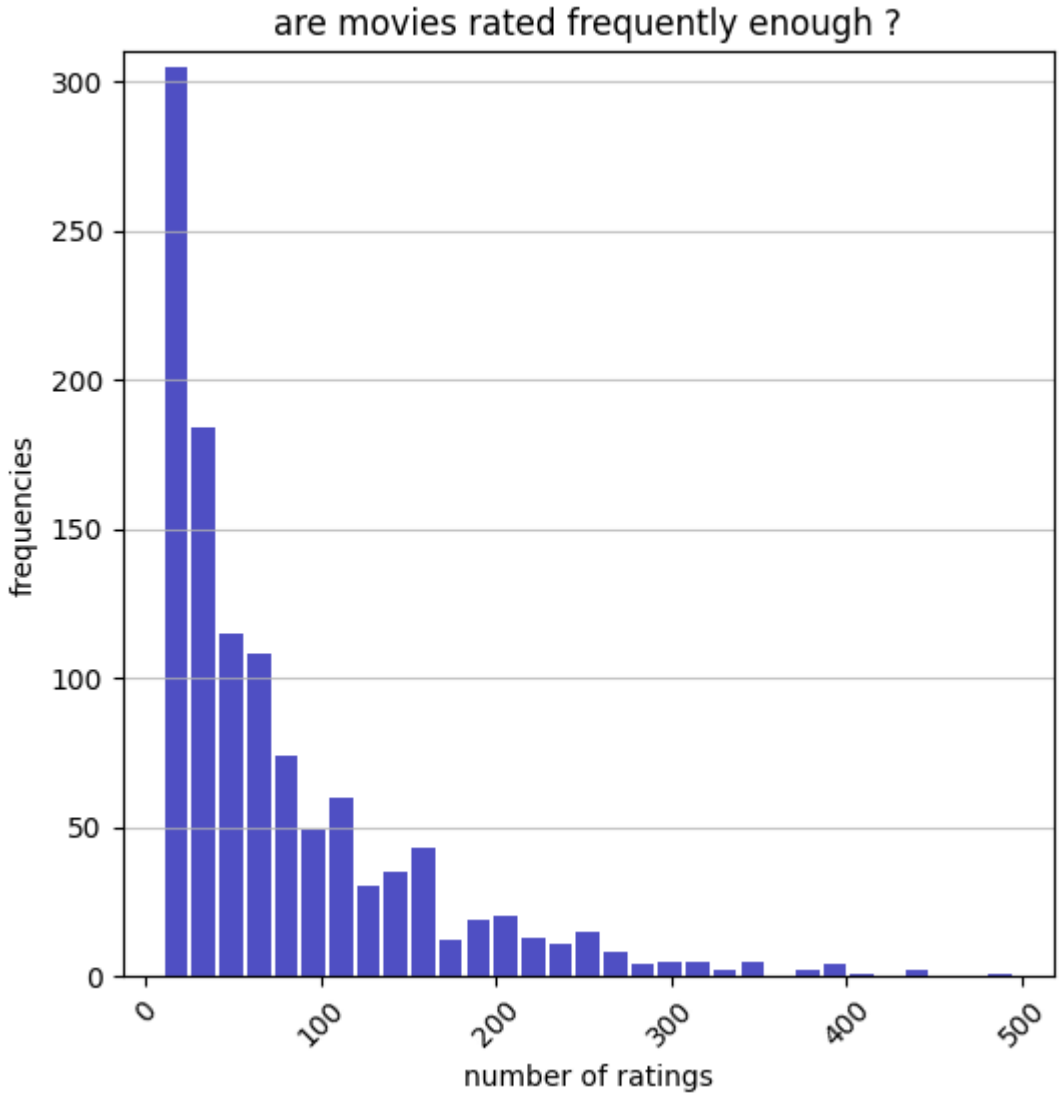
The final decision was to discard the geographical information as it might require extensive processing while displaying little to no statistical significance.

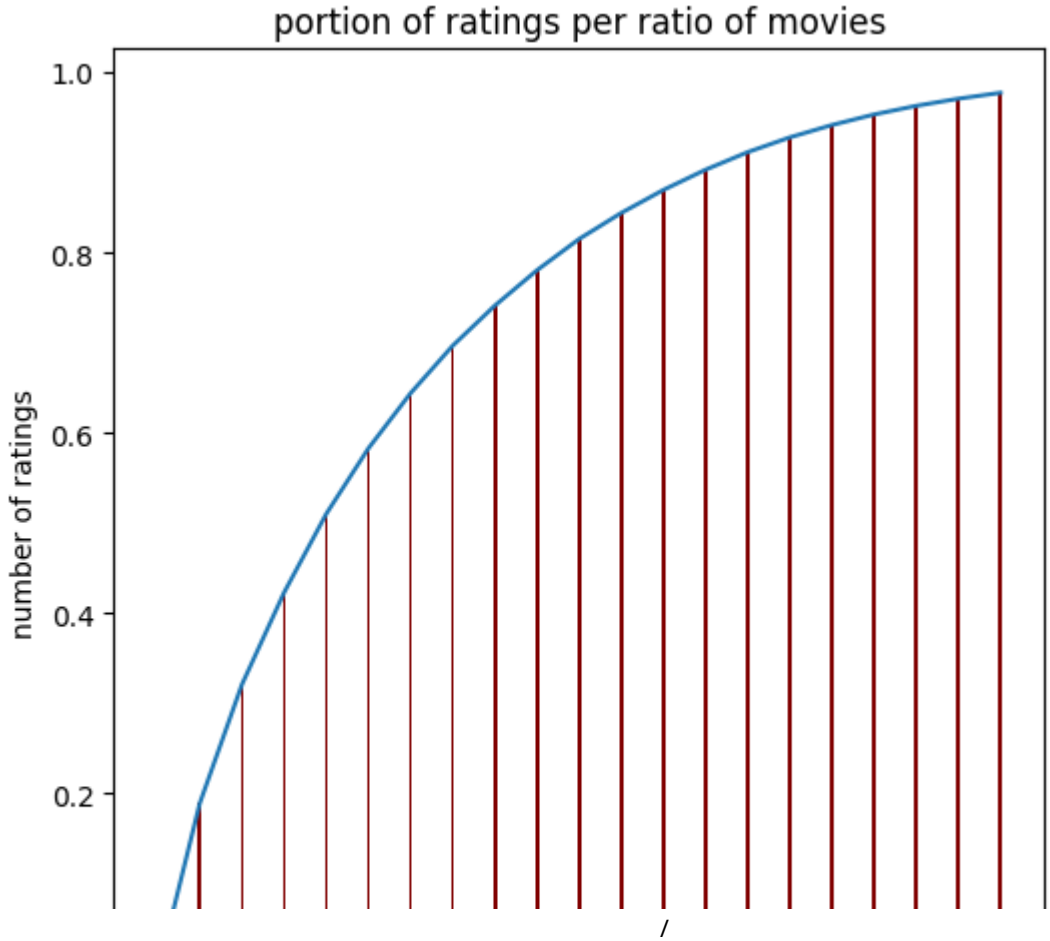
The initial distirbution of 'jobs' is quite similar to that of states. However, grouping jobs seems much more promising / natural than grouping users based on their geolocation information.

This hypothesis was further investigated while exploring the ratings data.

Ratings

The main remark while exploring the ratings, is the significant skewness of the distribution of the number of ratings. We can see that most of the movies have rated very few times, while a minority of movies have been rated frequently enough to build a statistically reliable profile of such movies.


are movies rated frequently enough ?
frequencies
number of ratings
0
50
100
150
200
250
300
0
100
200
300
400
500


portion of ratings per ratio of movies
number of ratings
0.2
0.4
0.6
0.8
1.0

Feature Engineering

The pull_data_together notebook includes a couple of feature engineering tricks to reprsent the 'job' field numerically: mainly in terms of ratings for specific genres. Nevertheless, it seems that the 'job' as well as 'genres' are not as discriminative since the representations of different jobs are quite similar as shown below: The values presented are cosine similarities

The final representation was produced by extracting the 'count', 'mean' and 'std' of the ratings of each job per genre. This representation expanded the user representation by $19 * 3 = 57$ features.

Model Implementation

Thought process

Based on the EDA carried out previously leads a number of conclusions:

1. The representation of the movie data is not expressive enough to build a content-based Recommender system

2. classical ML might not be enough since the interaction between the features is quite complex. Thus, Deep Learning presents itself as the most promising direction as it can learn and capture the complex and non-linear interactions between the features.

A brief literature review exposed me to very interesting ideas:

1. Collaborative filtering can be introduced in DL by using classification and negative simpling while learning embeddings of users and items

$$p(\mathcal{Y}, \mathcal{Y}^- | \mathbf{P}, \mathbf{Q}, \Theta_f) = \prod_{(u,i) \in \mathcal{Y}} \hat{y}_{ui} \prod_{(u,j) \in \mathcal{Y}^-} (1 - \hat{y}_{uj}). \quad (6)$$

Taking the negative logarithm of the likelihood, we reach

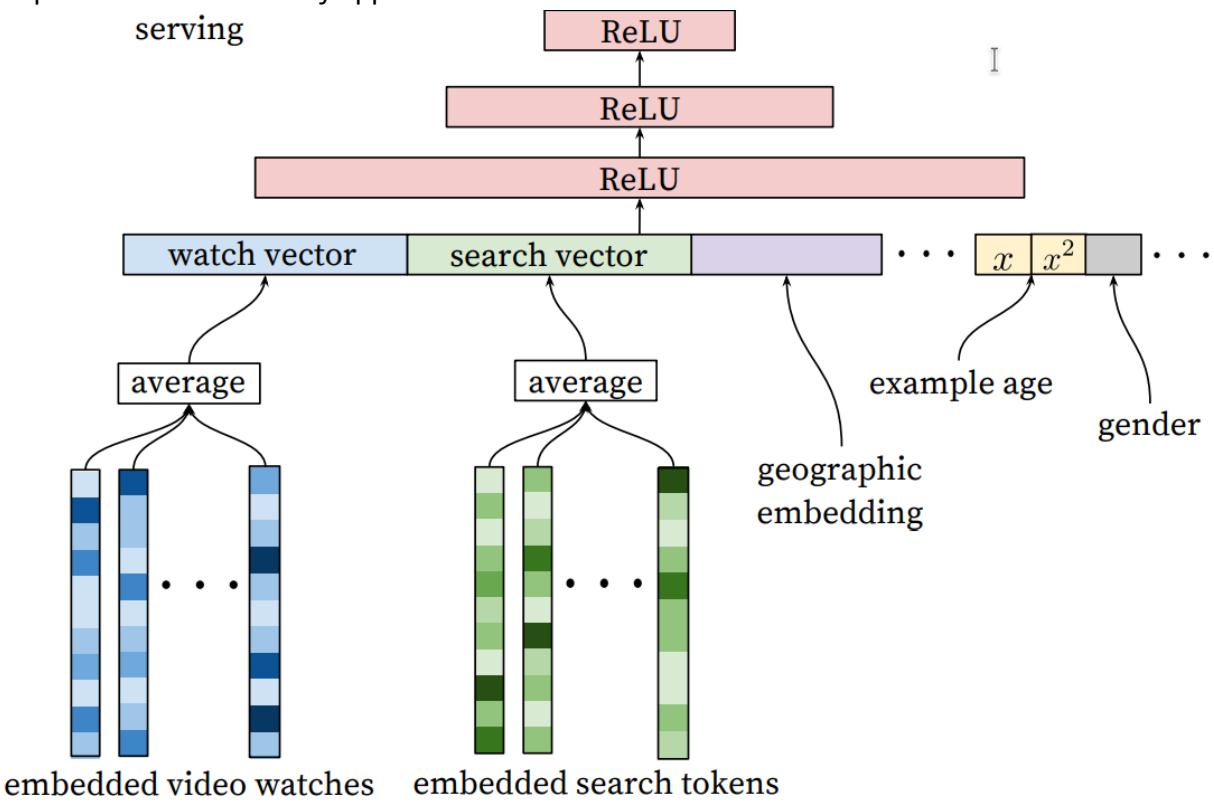
$$\begin{aligned} L &= - \sum_{(u,i) \in \mathcal{Y}} \log \hat{y}_{ui} - \sum_{(u,j) \in \mathcal{Y}^-} \log(1 - \hat{y}_{uj}) \\ &= - \sum_{(u,i) \in \mathcal{Y} \cup \mathcal{Y}^-} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}). \end{aligned} \quad (7)$$

This is the objective function to minimize for the NCF methods, and its optimization can be done by performing stochastic gradient descent (SGD). Careful readers might have realized that it is the same as the *binary cross-entropy loss*, also

The paper can be accessed through [1]

2. Youtube Research team managed to boost the performance of their Recommenders by aggregating the user's history: embedding of previous videos watched as well as search history. I tried to

incorporate this idea into my approach:



The paper can be accessed through [2]

Thus,

My suggested model has 4 main components:

1. 2 embeddings layers (nn.Embedding from pytorch). The model will be given the index of the user i and the index of the movie j. The model learns embeddings for both i and j. I denote the embedding dimension by n
2. A linear block (several dense layers + ReLU) that accepts the context vector, a vector of all the information about the user, the video, and the user history all concatenated into a single input vector. The linear block outputs a non-linear latent representation of the initial input of dimension $2 \cdot n$
3. A concatenation layer that concatenates the embedding of i-th user and j-th user and the output of the linear block: a $4 \cdot n$ vector
4. Another linear block that ends with 2 heads: One head of classification: whether the $i-th$ user watched the j-th movie and another for regression: predicting the user's rating.

Training Process

We can consider 2 main points in the training process:

1. I used the 'u1.base' and 'u1.test' for training. This split is initially provided in the data. It satisfies a common assumption that all users in test are present in training (while the items in test might be necessary seen in the train split)

2. Negative Sampling: I introduce negative sampling by passing both positive and negative pairs (user_id, item_id). The model learns by predicting both whether the user 'u_id' watched the movie 'i_id' and in the same time predict the rate for positive samples.

3. Adding history Data: The input to the model is of the following structure:

user_id, item_id, user features, user_history, video features

user_history is average of the video features weighted by their ratings for positive samples and the entire history for negative ones. Additionally, I sort the user data by the rating time (for a pair (u_id, i_id), the history will be the average the features of the movies that were rated before the pair in question)

Model Advantages and Disadvantages

Let's start with the disadvantages:

- This model is experimental. It is a combination of several ideas I encountered in the literature, online tutorials as explained above
- The model is relatively complex and might be prone to overfitting (mainly due to the small size of the dataset)
- Using the video information helps reduce the cold start problem. Nevertheless, the effect of the default embeddings (for our of vocabulary indices) is not exactly known

As for the advantages:

1. The model is not only trained for the classification but also for regression. So the model will not only learn to predict whether the user i is going to watch the movie j but also use the ratings (when available) to improve the model confidence
2. The model takes advantage of all the available data, which might help overcome the issue with the data size
3. The model uses negative sampling which helps avoids data folding.
4. Deep Learning models are much more likely to construct a complex and non-linear features that linear classifier simply cannot. This is crucial for our setting since data analysis shows that the interaction between the initial feature and the target (either rating or binary target) are quite complex.

Evaluation

Recommendataion Procedure

Given a pair (user: i, item: j), the model is trained to predict:

1. Whether i watched j
2. The rating that i gives to j (if watched)

The final objective is to recommend items to a given movie. Given the nature of the model, this objective can be achieved in 2 ways:

- for a user i, Predict the ratings for each pair (i, x) for each movie x in the test set, choose k movies with the top predicted ratings
- for a user i, Predict the probabilities that user i watched the movie x. Choose the movies with the highest probabilities.

Evaluation metrics

I evaluate the performance of the model by computing 4 metrics.

1. Mean Squared Error between the predicted ratings and the actual ones
2. Recall at k (R@k):
3. Precision at k (P@k):
4. Mean Average Precision

The last metrics are considered standard in MAchine Learing and related areas such as Information retrieval.

k was chosen as 20

Results

The model achieved the following results

1. MSE Loss on test split: 1.12. (common between the 2 recommendation approaches)

As for the Regression-based approach:

MAP: 0.005710814315827257

Precision@K: 0.0230246396790885

Recall@K: 0.021433214982249572

As for the Classification-based recommendation:

MAP: 0.0053081095561827545

Precision@K: 0.022287930256884646

Recall@K: 0.02070060509076555

We can see that both approaches perform poorly on the test split. (predicting 1 to 2 items out of the true items per user).

This can be explained by the quality of the data:

1. skewness in ratings: many users have few ratings overall (which is not enough to model their taste in movies) and most films are assigned few ratings
2. 19 binary very sparse features are definitely not enough to model such a complex concept such as a movie.

3. Little to no statistical significant for most features: genre, zip_code, job...